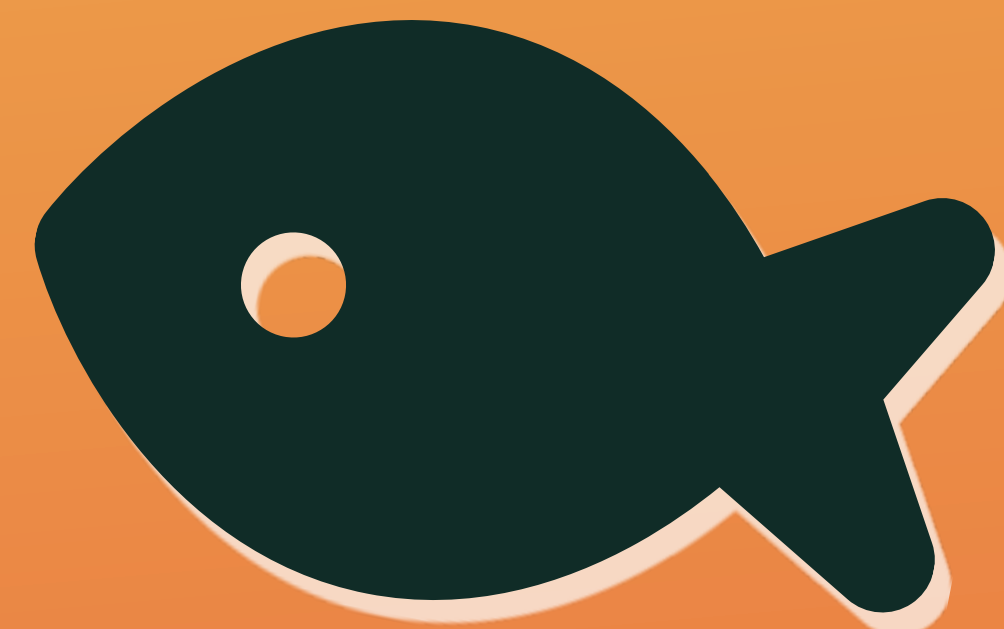
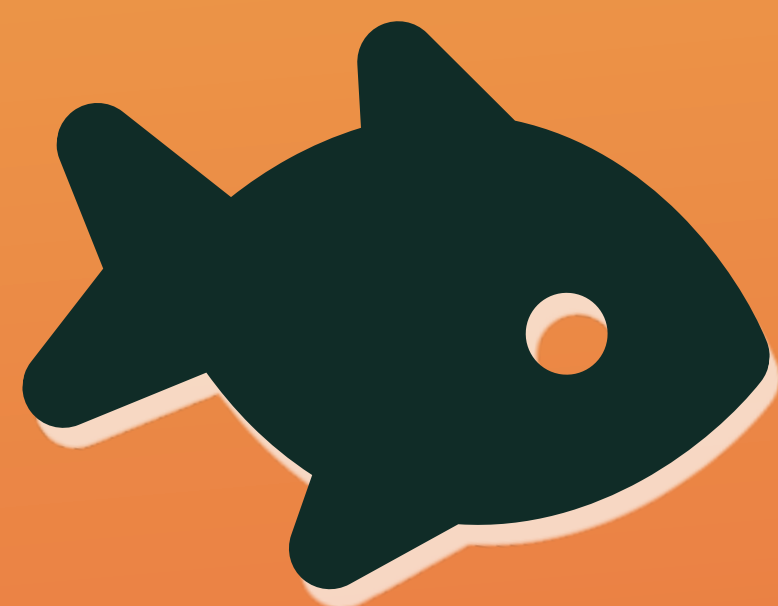
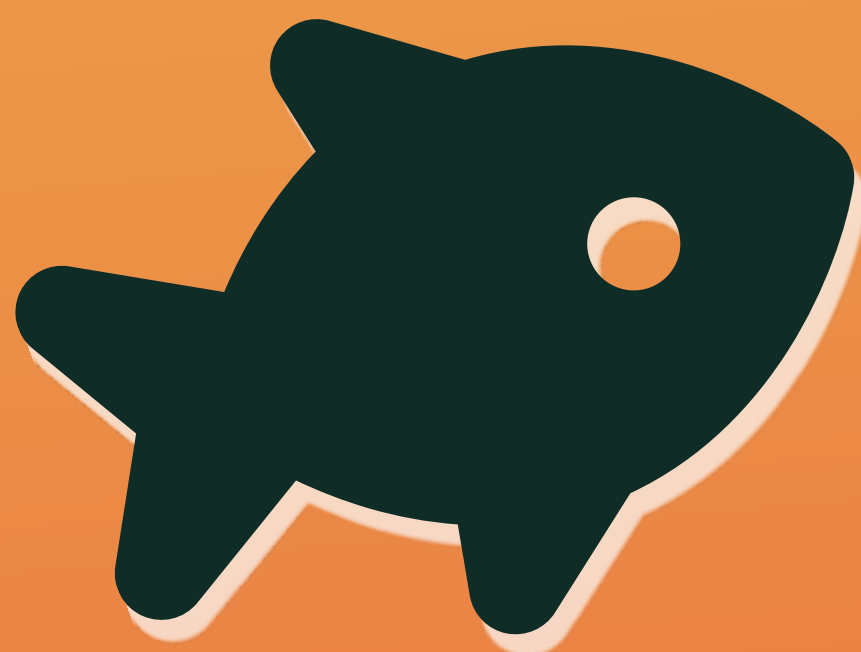
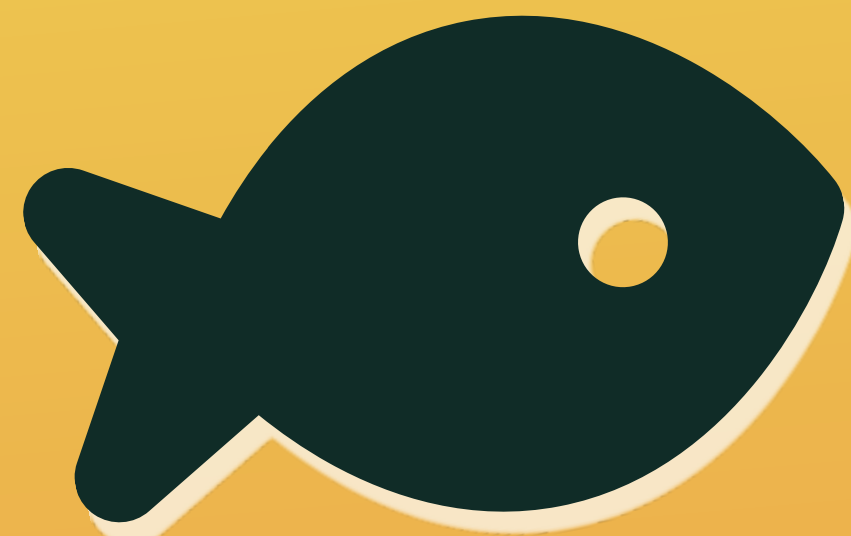
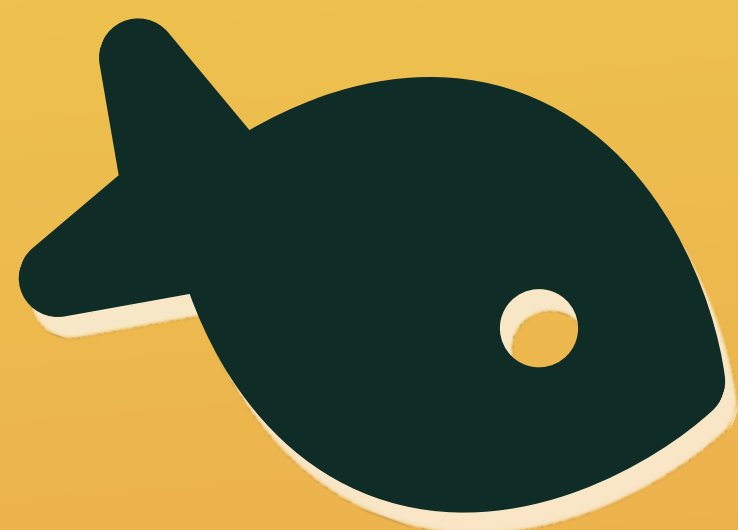
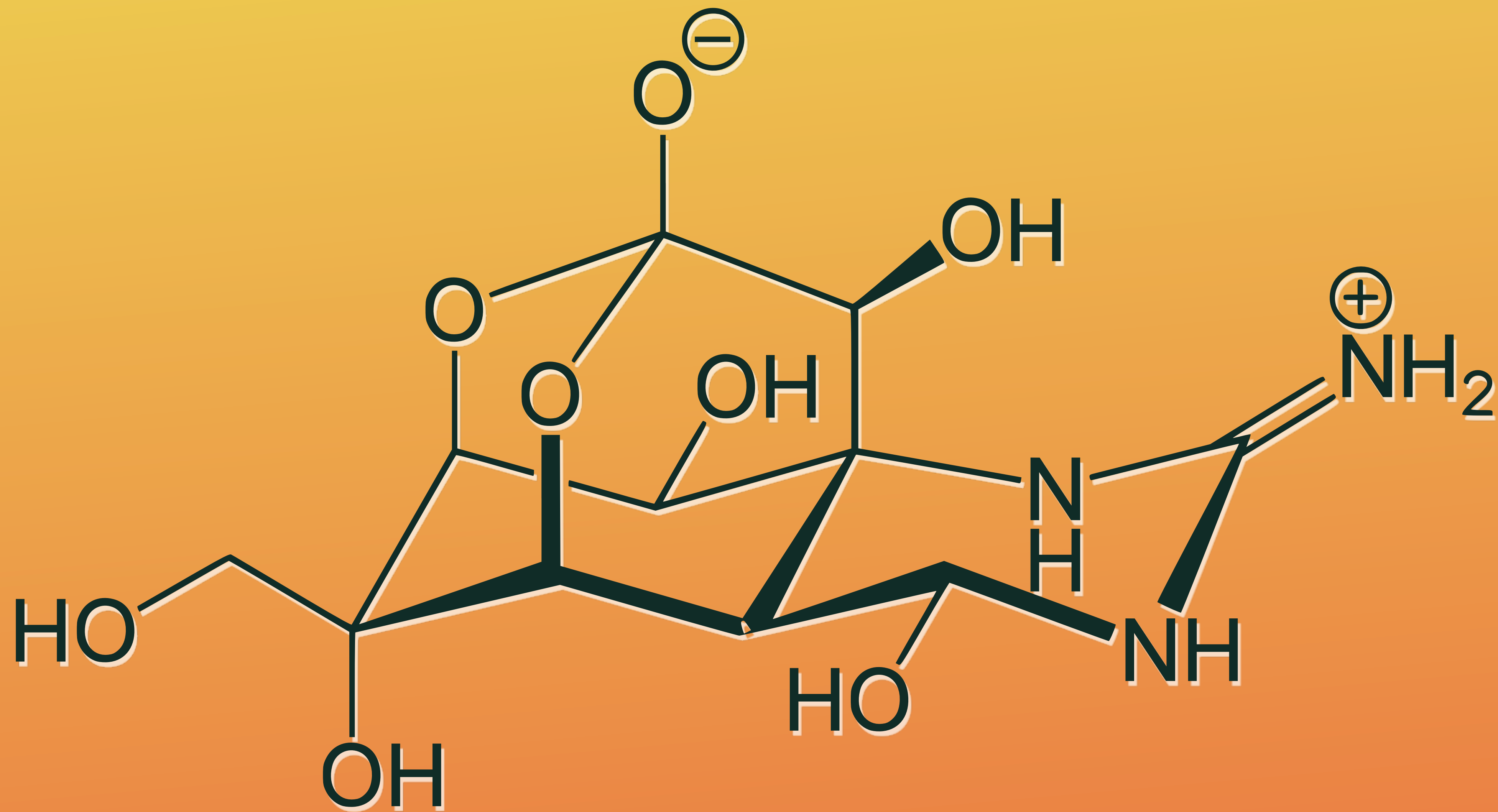


FUGU

POSSIBILITIES WITH WEB CAPABILITIES







TETRODOTOXIN



PROJECT FUGU 🐡

Google



Microsoft

intel

developer.chrome.com/docs/capabilities



PROJECT FUGU

Accelerometer API

Ambient Light Sensor API

Background Fetch API

Background Sync API

Badging API

Clipboard API

Compute Pressure API

Contact Picker API

Device Memory API

Device Posture API

EyeDropper API

File Handling API

File System Access API

File System Observer API

Font Access API

Gamepad API

Geolocation Sensor API

Gravity Sensor API

Idle Detection API

Fullscreen API

Magnetometer API

Periodic Background Sync API

Picture-in-Picture API

Presentation API

Screen Wake Lock API

Shape Detection API

Virtual Keyboard API

Wake Lock API

Web Audio API

Web Bluetooth API

Web MIDI API

Web NFC API

Web Serial API

Web Share API

Web Codecs API

Web HID API

Web GPU API

Web USB API

Web XR Device API

Window Placement API

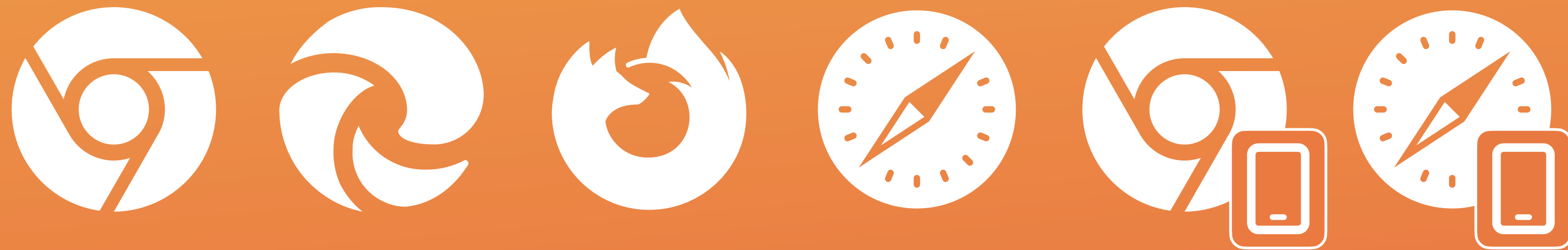


CLIPBOARD API



CLIPBOARD API

- **copy text, images, or other data formats to and from the clipboard**
- **requires a user gesture; for example, a button click**
- **pasting requires explicit permission from the user**



```
const copyTextToClipboard = async (text) => {  
  await navigator.clipboard.writeText(text);  
};
```



```
const copyTextToClipboard = async (text) => {  
  await navigator.clipboard.writeText(text);  
};  
  
const readTextFromClipboard = async () => {  
  const text = await navigator.clipboard.readText();  
  return text;  
};
```



```
const copyTextToClipboard = async (text) => {  
  await navigator.clipboard.writeText(text);  
};  
  
const readTextFromClipboard = async () => {  
  const text = await navigator.clipboard.readText();  
  return text;  
};  
  
const copyPngImageToClipboard = async (imagePath) => {  
  const response = await fetch(imagePath);  
  const blob = await response.blob();  
  const clipboardItem = new ClipboardItem({ "image/png": blob });  
  await navigator.clipboard.write([clipboardItem]);  
};
```



DEMO

CLIPBOARD API

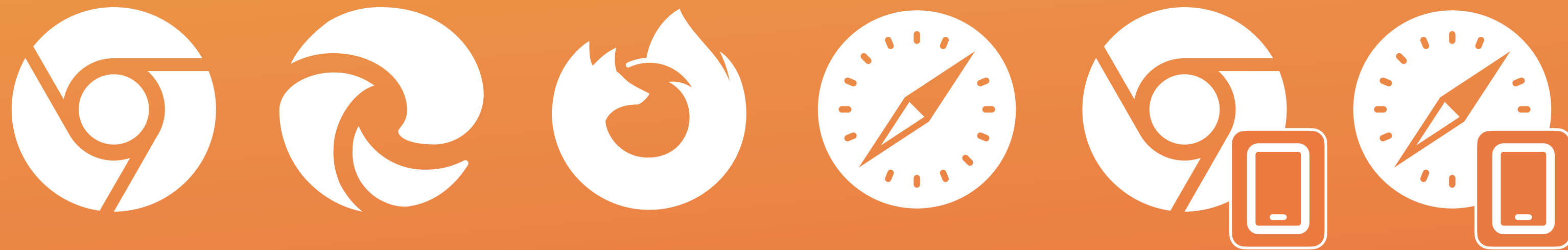


FULLSCREEN API



FULLSCREEN API

- **remove distractions like browser UI, so users can focus entirely on the content**
- **full document or a specific element can be displayed in full-screen mode**
- **requires a user gesture; for example, a button click**



⬤ ⬤ ⬤ (); script.js ✕

```
const openFullscreen = () => {  
  document.documentElement.requestFullscreen();  
};
```



```
const openFullscreen = () => {  
  document.documentElement.requestFullscreen();  
};  
  
const exitFullscreen = () => {  
  if (document.fullscreenElement) { // prevent error when not fullscreen  
    document.exitFullscreen();  
  }  
};
```



```
const openFullscreen = () => {
  document.documentElement.requestFullscreen();
};

const exitFullscreen = () => {
  if (document.fullscreenElement) { // prevent error when not fullscreen
    document.exitFullscreen();
  }
};

const video = document.querySelector("#video");

const playVideoFullscreen = (video) => {
  video.requestFullscreen();
  video.play();
};

// user can also exit with Esc key or a button in the video player
```



DEMO

FULLSCREEN API



PICTURE-IN-PICTURE API



PICTURE-IN-PICTURE API

- enable a floating, always-on-top frame
- commonly used for video playback, experimental for other content
- requires a user gesture; for example, a button click





() ; script.js



{ } ; style.css

< > index.html

() ; pip.js

```
const video = document.querySelector("#video");

const toggleVideoPictureInPicture = (video) => {
  if (document.pictureInPictureEnabled) {
    if (document.pictureInPictureElement) {
      document.exitPictureInPicture();
    } else {
      video.requestPictureInPicture();
    }
  }
};
```



rowdy.codes



rowdyrabouw



() ; script.js



{ } ; style.css

< > index.html

() ; pip.js

```
const video = document.querySelector("#video");

const toggleVideoPictureInPicture = (video) => {
  if (document.pictureInPictureEnabled) {
    if (document.pictureInPictureElement) {
      document.exitPictureInPicture();
    } else {
      video.requestPictureInPicture();
    }
  }
};

const toggle = document.querySelector("#toggle");

if (!document.pictureInPictureEnabled) {
  toggle.style.display = "none";
}
```



rowdy.codes



rowdyrabouw



() ; script.js

{ } ; style.css ✕

< > index.html

() ; pip.js

```
:picture-in-picture {  
  display: none;  
}
```





() ; script.js

{ } ; style.css

< > index.html X

() ; pip.js

```
<div id="pipContainer">
  <div id="pipContent">
    <h2>DocumentPictureInPicture</h2>
    <p>This interface allows you to create an always-on-top window.</p>
    <p>This is currently only supported in Chromium based browsers.</p>
  </div>
</div>
```



rowdy.codes



rowdyaboutw



() ; script.js

{ } ; style.css

< > index.html

() ; pip.js



```
const pipContainer = document.querySelector("#pipContainer");
const pipContent = document.querySelector("#pipContent");

const togglePipContent = async () => {

  const pipWindow = await window.documentPictureInPicture.requestWindow({
    width: 600, height: 400,
  });
  pipWindow.document.body.append(pipContent);

};
```



rowdy.codes



rowdyrabouw



() ; script.js

{ } ; style.css

< > index.html

() ; pip.js



```
const pipContainer = document.querySelector("#pipContainer");
const pipContent = document.querySelector("#pipContent");

const togglePipContent = async () => {

  const pipWindow = await window.documentPictureInPicture.requestWindow({
    width: 600, height: 400,
  });
  pipWindow.document.body.append(pipContent);

  pipWindow.addEventListener("pagehide", (event) => {
    pipContainer.append(pipContent);
  });
};
```



rowdy.codes



rowdyrabouw



(); script.js

{}; style.css

<> index.html

(); pip.js



```
const pipContainer = document.querySelector("#pipContainer");
const pipContent = document.querySelector("#pipContent");

const togglePipContent = async () => {
  if (window.documentPictureInPicture.window) {
    pipContainer.append(pipContent);
    window.documentPictureInPicture.window.close();
    return;
  }
  const pipWindow = await window.documentPictureInPicture.requestWindow({
    width: 600, height: 400,
  });
  pipWindow.document.body.append(pipContent);

  pipWindow.addEventListener("pagehide", (event) => {
    pipContainer.append(pipContent);
  });
};
```



rowdy.codes



rowdyrabouw

DEMO

PICTURE-IN-PICTURE API



WEB SHARE API



WEB SHARE API

- share content directly with native sharing capabilities of the user's device
- sharing options depend on the apps installed on the user's device
- limitations on the types of content (audio, video, images, text and pdf)
- requires a user gesture; for example, a button click



```
const shareUrl = async () => {  
  if (navigator.canShare) {  
    await navigator.share({  
      title: "Check out this website!", // may be ignored by the target  
      text: "I found this website about project Fugu you should see.",  
      url: "https://webcapabilities.nl",  
    });  
  }  
};
```



```
const shareFile = async () => {  
  if (navigator.canShare) {  
    const file = new File(["Web Share API", "\n\nThe Web Share API allows  
web applications to share content directly with native sharing  
capabilities of the user's device."])  
  
  };  
  
}  
};
```



```
const shareFile = async () => {  
  if (navigator.canShare) {  
    const file = new File(["Web Share API", "\n\nThe Web Share API allows  
web applications to share content directly with native sharing  
capabilities of the user's device."], "fugu.txt"  
  );  
  }  
};
```



```
const shareFile = async () => {  
  if (navigator.canShare) {  
    const file = new File(["Web Share API", "\n\nThe Web Share API allows  
web applications to share content directly with native sharing  
capabilities of the user's device."], "fugu.txt",  
      { type: "text/plain" }  
    );  
  }  
};
```



```
const shareFile = async () => {  
  
  if (navigator.canShare) {  
    const file = new File(["Web Share API", "\n\nThe Web Share API allows  
web applications to share content directly with native sharing  
capabilities of the user's device."], "fugu.txt",  
      { type: "text/plain"}  
  );  
  
  await navigator.share({  
    files: [file], // array of files to share  
    title: "Web Share API",  
    text: "Check out the information in this file!",  
  });  
}  
  
};
```



```
const shareFile = async () => {  
  
  if (navigator.canShare) {  
    const response = await fetch("fugu.pdf");  
    const blob = await response.blob();  
    const file = new File([blob], "fugu.pdf",  
                          { type: "application/pdf" });  
  
    await navigator.share({  
      files: [file], // array of files to share  
      title: "Web Share API",  
      text: "Check out the information in this file!",  
    });  
  }  
  
};
```



DEMO

WEB SHARE API



WEB USB API



WEB USB API

- **communicate directly with USB devices**
- **connecting requires explicit permission from the user**
- **requires a user gesture; for example, a button click**



⬤ ⬢ ⬤ (); script.js ✕

```
const connectAndsendDataToUsbDevice = async () => {  
  // check if the Web USB API is supported  
  if (navigator.usb) {
```

```
  }  
}
```



⬤ ⬢ ⬤ (); script.js ✕

```
const connectAndsendDataToUsbDevice = async () => {  
  // check if the Web USB API is supported  
  if (navigator.usb) {  
    // select a USB device (hexidecimal values for vendorId and productId)  
    const device = await navigator.usb.requestDevice(  
      { filters: [{ vendorId: 0x0416, productId: 0x5011 }] });  
  
  }  
}
```



```
const connectAndsendDataToUsbDevice = async () => {  
  // check if the Web USB API is supported  
  if (navigator.usb) {  
    // select a USB device (hexidecimal values for vendorId and productId)  
    const device = await navigator.usb.requestDevice(  
      { filters: [{ vendorId: 0x0416, productId: 0x5011 }] });  
    // establish a connection with the USB device  
    await device.open();  
  
  }  
}
```



```
const connectAndsendDataToUsbDevice = async () => {  
  // check if the Web USB API is supported  
  if (navigator.usb) {  
    // select a USB device (hexidecimal values for vendorId and productId)  
    const device = await navigator.usb.requestDevice(  
      { filters: [{ vendorId: 0x0416, productId: 0x5011 }] });  
    // establish a connection with the USB device  
    await device.open();  
    // select a configuration to prepare the device for communication  
    await device.selectConfiguration(1);  
  }  
}
```



```
const connectAndsendDataToUsbDevice = async () => {  
  // check if the Web USB API is supported  
  if (navigator.usb) {  
    // select a USB device (hexidecimal values for vendorId and productId)  
    const device = await navigator.usb.requestDevice(  
      { filters: [{ vendorId: 0x0416, productId: 0x5011 }] });  
    // establish a connection with the USB device  
    await device.open();  
    // select a configuration to prepare the device for communication  
    await device.selectConfiguration(1);  
    // claim an interface to perform data transfers  
    await device.claimInterface(0);  
  }  
}
```



```
const connectAndsendDataToUsbDevice = async () => {
  // check if the Web USB API is supported
  if (navigator.usb) {
    // select a USB device (hexidecimal values for vendorId and productId)
    const device = await navigator.usb.requestDevice(
      { filters: [{ vendorId: 0x0416, productId: 0x5011 }] });
    // establish a connection with the USB device
    await device.open();
    // select a configuration to prepare the device for communication
    await device.selectConfiguration(1);
    // claim an interface to perform data transfers
    await device.claimInterface(0);
    // send data to the device
    // the first parameter is the endpoint number
    // the second parameter is the data to be sent as a Uint8Array
    await device.transferOut(1, new Uint8Array([0x01, 0x02, 0x03]));
  }
}
```





DEMO

WEB USB API



WIN A FUGU DOG TOY



Send your best tech joke to my printer!



DEMO

WEB USB API



FILE SYSTEM API



FILE SYSTEM API

- **interact with files on the user's local device**
- **Adobe Photoshop Web**
- **vscode.dev**
- **requires a user gesture; for example, a button click**



```
const saveFile = async () => {  
  const fileHandle = await window.showSaveFilePicker({  
    types: [  
      {  
        accept: { 'text/plain': ['.txt'] }  
      }  
    ]  
  });  
  const writable = await fileHandle.createWritable();  
  await writable.write('Hello World!');  
  await writable.close();  
};
```



```
const readFile = async () => {  
  const fileHandle = await window.showOpenFilePicker(  
    {  
      types: [  
        {  
          accept: { 'text/plain': ['.txt'] }  
        }  
      ]  
    })  
  );  
  const file = await fileHandle[0].getFile();  
  const contents = await file.text();  
  console.log(contents);  
};
```



DEMO

FILE SYSTEM API



WIN A FUGU DOG TOY



Send your best tech joke to my printer!





DANKE SCHÖN



ROWDY.CODES/WAD

